INFORMATION AND CONTROL 27, 374–377 (1975)

A Note on Cycle Grammars

AZRIEL ROSENFELD

Computer Science Center, University of Maryland, College Park, Maryland 20742

Grammars whose languages consist of cycles ("necklaces") rather than strings are considered. If G is context free, and we regard G as generating cycles instead of strings, the resulting language is just what we would get if we "bent" the strings of $L(G)$ into cycles. This is no longer true if G is context sensitive. However, in this case too, the context-sensitive cycle languages are just the "bendings" of the context-sensitive string languages. Automata on cyclic tapes are also discussed.

About ten years ago a grammar was given (Ledley, 1964) for outlines of chromosome shapes, regarded as sequences of basic shape elements (straight segments, sharp convexities, sharp concavities, etc.). Such grammars have since been extensively studied by Fu (e.g., Huang and Fu, 1972). Recently (Siromoney and Siromoney, 1974), "polar-coordinate" matrix grammars were used to generate patterns having various types of rotational symmetry.

The languages of these grammars should, strictly speaking, consist of "necklaces" (rather than strings) of symbols—i.e., of *cyclically ordered* sequences of symbols. Of course, one can produce a "necklace" by allowing a grammar to generate a string, and then "bending" the string until it closes on itself. A more natural alternative would be to regard the rules of the grammar as operating on sentential forms that are themselves cycles rather than strings. If the grammar is context sensitive, the languages generated by it in these two ways are not necessarily the same. However, as we shall see below, the classes of languages that can be generated in the two ways are identical.

Let us first suppose that the grammar G is context free, so that its rules have only single symbols on their left hand sides. In this case it cannot matter whether we regard the sentential forms of G as strings while we are generating a sentence, and only bend them into cycles when we are done, or whether we regard them as cycles all along, since even if we do the latter,

we cannot benefit from the fact that the tails of these strings are now adjacent to their heads.

We can express this observation very concisely if we introduce the following notation: Let $L(G)$ be the (string) language generated by G, and let $\tilde{L}(G)$ be the language generated when sentential forms are regarded as cycles. Also, for any string language L, let $\hat{L}$ be the set of cycles that results from bending the strings of L. Then if G is context free, we have $\tilde{L}(G) = \widehat{L(G)}$.

In the context sensitive case, this is no longer true. However, we can show that for any grammar G, there exist grammars G_1 and G_2 such that

$$\tilde{L}(G) = \widehat{L(G_1)} \qquad \text{and} \qquad \widehat{L(G)} = \tilde{L}(G_2)$$

Moreover, if G is context sensitive, so are G_1 and G_2.

To prove the first part, let $A_1 \cdots A_m \to B_1 \cdots B_n$ be any rule of G. We give G_1 this rule together with a set of rules of the form $\#A_{i+1} \cdots A_m \to \#C_i$, for $1 \leqslant i < m$. If any of these rules applies, we can only get rid of the C_i by having it move to the right-hand end of the string and then using a rule of the form $A_1 \cdots A_i C_i \# \to B_1 \cdots B_n \#$. If we want to avoid excessive blocking, we can make the creation and movement of the C_i's reversible. Note that if the rules of G never decrease the length of a string, neither do the rules of G_1.

For the second part, we need only design G_2 so that no rule is ever able to cross over some marked point in the cycles (corresponding to the ends of the strings being generated by G). To this end, we can regard the initial symbol of G_2 as "primed"; and for each rule $A_1 \cdots A_m \to B_1 \cdots B_n$ of G, we give G_2 this rule together with the rule $A_1 \cdots A_m' \to B_1 \cdots B_n'$. Here the prime marks the last symbol of the string whose derivation G_2 is simulating. If G has no erasing rules, it is clear that any sentential form of G_2 has exactly one primed symbol. While simulating G-derivations in G_2, we represent the terminal symbols of G by "pseudoterminals" in G_2, which can be turned into real terminals by rules of the form $T \to t$ and $T' \to t$. Note that if we do this before the simulation is finished, we may block, since the rules of G_2 that imitate those of G do not involve real terminals. In particular, even if we turn the primed symbol into a terminal too soon, it is still impossible for any rule to cross over that symbol, since such a rule would have to involve a real terminal.

If G has erasing rules, we cannot use the scheme just described, since there is no place to put the prime in a null right-hand side. However, we can instead begin with a rule which turns the initial symbol into $S\natural$, say. Since the rules of G do not involve $\natural$, we can safely use them in G_2 without any possibility

of destroying the ♮ or crossing its location. Here again, we should use pseudoterminals in G_2, and terminate using rules of the form $\natural T \rightarrow t$ and $tU \rightarrow tu$; this is safe by the remarks at the end of the preceding paragraph.

In conclusion, we consider the related question of acceptance of our cycle languages by automata that have cyclical input tapes. We can formulate this question more precisely as follows: let L be a string language, and let $\hat{L}$ be the cycle language obtained by bending the strings of L, as above. We are interested in the acceptance of $\hat{L}$ by an automaton, which we allow to start from the point of its input cycle corresponding to the left end of the string. The automaton can move either clockwise or counterclockwise, and can tell whichis which (i.e., our cycles are all directed cycles). Note that since a cycle has no ends, the automaton is tape-bounded.

Let $\mathscr{M}$ be any class of automata, e.g., finite state, pushdown, etc., which are allowed to move in both directions. Given a cycle automaton $M^0 \in \mathscr{M}$, we can easily design a string automaton $M' \in \mathscr{M}$ that imitates M^0. In fact, when M' reaches an end of its input string, and M^0 would have moved beyond that point, M' simply remembers its state, keeps its auxiliary storage intact, moves to the other end of the string, and resumes the imitation. It follows that if $\mathscr{L}^0$ is the class of cycle languages, and $\mathscr{L}$ the class of string languages, accepted by $\mathscr{M}$, then we have $\mathscr{L}^0 \subseteq \{\hat{L} \mid L \in \mathscr{L}\}$.

Conversely, let $\mathscr{M}$ be any class of automata, and let $\mathscr{M}_p$ be the class of automata with the same power (writing ability, auxiliary storage, etc.) plus one additional pebble. Given a string automaton $M \in \mathscr{M}$, we can easily design a cycle automaton $M_p \in \mathscr{M}_p$ that imitates M. Indeed, M_p need only begin by marking its starting point with the pebble; it can now imitate M's behavior on a string, using the pebble to identify the ends of the string, and never moving past the pebble. It follows that if $\mathscr{L}$ is the class of string languages accepted by $\mathscr{M}$, and $\mathscr{L}_p$ the class of cycle languages accepted by $\mathscr{M}_p$, then we have $\mathscr{L}_p \supseteq \{\hat{L} \mid L \in \mathscr{L}\}$.

If we take $\mathscr{M}$ to be the class of LBA's, we have $\mathscr{M}_p = \mathscr{M}$—adding a pebble gives no additional power. It follows from the last two paragraphs that the class of cycle languages accepted by LBA's is exactly the class of context-sensitive cycle languages.

For finite-state automata, on the other hand, the class of cycle languages that they accept is much smaller than the finite-state cycle languages. For example, suppose that M is an FSA that accepts the cycle $\rho = A_1 \cdots A_n$. It is clear that this must happen within a bounded number N of time steps, where N is a function of n and of M's state set size. Now consider the cycle $\tau = \rho^{2N+1}\sigma$, where σ is an arbitrary string. If we start M from the middle copy of ρ, it must also accept τ, since it can never get far enough away

from its starting point to tell that it is not on ρ. We have thus shown that if M accepts any cycle ρ, it also accepts an infinite set of cycles of the form $\rho^{2N+1}\sigma$. In particular, the finite cycle languages are not finite-state.

Acknowledgment

The support of the Information Systems Branch, Office of Naval Research, under Contract N00014-67A-0239-0012, is gratefully acknowledged.

Received: June 19, 1974

References

Huang, T., and Fu, K. S. (1972). Stochastic syntactic analysis for programmed grammars and syntactic pattern recognition, *Computer Graphics and Image Processing* **1**, 257–283.

Ledley, R. S. (1964). High-speed automatic analysis of biomedical pictures, *Science* **146**, 216–223.

Siromoney, G., and Siromoney, R. (1974). Radial grammars and biological systems, "Proceedings of the Conference on Biologically Motivated Automata Theory," IEEE Publication 74CH0-889-6C, 92–96.